

# Relating Knowledge and Coordinated Action: The Knowledge of Preconditions Principle

Yoram Moses\*

Technion—Israel Institute of Technology  
moses@ee.technion.ac.il

The *Knowledge of Preconditions* principle (**KoP**) is proposed as a widely applicable connection between knowledge and action in multi-agent systems. Roughly speaking, it asserts that if some condition  $\varphi$  is a necessary condition for performing a given action  $\alpha$ , then *knowing*  $\varphi$  is also a necessary condition for performing  $\alpha$ . Since the specifications of tasks often involve necessary conditions for actions, the **KoP** principle shows that such specifications induce knowledge preconditions for the actions. Distributed protocols or multi-agent plans that satisfy the specifications must ensure that this knowledge be attained, and that it is detected by the agents as a condition for action. The knowledge of preconditions principle is formalised in the runs and systems framework, and is proven to hold in a wide class of settings. Well-known connections between knowledge and coordinated action are extended and shown to derive directly from the **KoP** principle: a *common knowledge of preconditions* principle is established showing that common knowledge is a necessary condition for performing simultaneous actions, and a *nested knowledge of preconditions* principle is proven, showing that coordinating actions to be performed in linear temporal order requires a corresponding form of nested knowledge.

**Keywords:** Knowledge, multi-agent systems, common knowledge, nested knowledge, coordinated action, knowledge of preconditions principle.

## 1 Introduction

While epistemology, the study of knowledge, has been a topic of interest in philosophical circles for centuries and perhaps even millennia, in the last half century it has seen a flurry of activity and applications in other fields such as AI [19], game theory [2] and distributed computing [13]. At least in the latter two fields a particular, information-based, notion of knowledge plays a prominent and useful role.

This paper proposes an essential connection between knowledge and action in such a setting. Using  $\text{does}_i(\alpha)$  to denote “Agent  $i$  is performing action  $\alpha$ ” and  $K_i\varphi$  to denote that Agent  $i$  knows the fact  $\varphi$ , the connection can intuitively be formulated as follows:

### The KNOWLEDGE OF PRECONDITIONS Principle (**KoP**):

If  $\varphi$  is a necessary condition for  $\text{does}_i(\alpha)$   
then  $K_i\varphi$  is a necessary condition for  $\text{does}_i(\alpha)$

This statement appears deceptively simple. In fact, many successful applications of knowledge to the design and analysis of distributed protocols over the last three decades are rooted in the **KoP**. Moreover, some of the deeper insights obtained by knowledge theory in this field can be derived in a fairly direct

---

\*The Israel Pollak academic chair at Technion. This work was supported in part by ISF grant 1520/11.

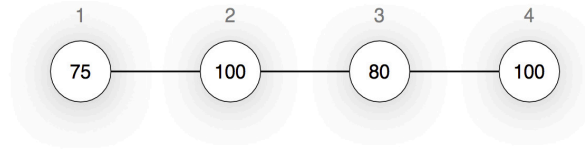


Figure 1: A simple four-agent system

fashion from the **KoP**. We will argue and demonstrate that this principle lies at the heart of coordination in many distributed and multi-agent systems.

This paper is structured as follows. Section 1.1 illustrates the central role of knowledge in a natural distributed systems application. Section 1.2 provides a high-level discussion of the knowledge of preconditions principle and its connection to coordinating actions. In Section 2 we review and discuss the modelling of knowledge in the runs and systems model of distributed systems based on [10]. A formal statement and proof of the **KoP** are presented in Section 3. Then, in Section 4, the **KoP** is used to establish a *common knowledge of preconditions* principle. It states that in order to perform simultaneously coordinated actions, agents must first attain common knowledge of any of the actions' preconditions. An example of its use is provided in Section 4.1. Section 5 present an additional use of the **KoP**, and shows that coordinating a sequence of actions to occur in a prescribed temporal order requires attaining nested knowledge of their preconditions. Finally, Section 6 discusses additional applications, extensions and future directions.

### 1.1 The Case for Knowledge in Distributed Systems

Why should knowledge play a central role in distributed computing? As pointed out in [13], most everyone who designs or even just tries to study the workings of a distributed protocol is quickly found talking in terms of knowledge, making statements such as “*once the process receives an acknowledgement, it knows that the other process is ready...*”. An essential aspect of distributed systems is the fact that an agent chooses which action to perform based on the local information available to it, which typically provides only a partial view of the overall state of the system. To get a sense of the role of knowledge in distributed systems, consider the following example.

**Example 1.** *Given is a distributed network modeled by a graph, with agents located at the nodes, and the edges standing for communication channels (see Figure 1). In the problem we shall call **Computing the Max** (or CTM for short), each agent  $i$  starts out with a natural number  $v_i \in \mathbb{N}$  as an initial value. The goal is to have Agent 1 print the maximum of all of the initial values (we denote this value by  $\text{Max}$ ), and print nothing else. In the instance depicted in Figure 1, the maximal value happens to be 100. Initially, Agent 1 clearly can't print its own initial value of 75. Suppose that Agent 1 receives a message  $\mu \triangleq "v_2 = 100"$  from Agent 2 reporting that its value is 100. At this point, Agent 1 has access to the maximum, and printing 100 would satisfy the problem specification. Compare this with a setting that is the same in all respects, except that Agent 3's value is  $v_3 = 150$ . In this case, of course,  $\text{Max} \neq 100$  and so printing 100 is forbidden. But if Agent 1 can receive the same message  $\mu$  under similar circumstances in both scenarios,*

then it is unable to distinguish whether or not  $\text{Max} = 100$  upon receiving  $\mu$ . Intuitively, even in the first scenario, the agent does not know that  $\text{Max} = 100$ .

What information does Agent 1 need, then, in order to be able to print the maximum? Notice that it is not necessary, in general, to collect all of the initial values in order to print the maximum. For example, suppose that the agents follow a bottom-up protocol in which values are sent from right to left, starting from Agent 4, and every agent passes to the left the larger of its own value and the value it received from its neighbor on the right (if such a neighbor exists). In this protocol, Agent 1 can clearly print the maximum after receiving the message " $v_2 = 100$ ", and seeing just one value besides its own. Interestingly, even collecting all of the values is not a sufficient condition for printing the Max. Imagine a setting in which the network is as in Figure 1, but Agent 1 considers it possible that there are more than four nodes in the network. In this case, even if Agent 1 receives all (four) values, it may still need to wait for proof that there is no additional, larger, value in the system.  $\square$

CTM is a simplified example in the spirit of many distributed systems applications. In fact, a central problem called *Leader Election*, for example, is often solved by computing a node with maximal ID [1, 17]. The solution to such a problem is typically in the form of a set of short computer programs (jointly constituting a *distributed protocol*), each executed at one of the nodes. When the nodes follow such a protocol, the resulting execution should satisfy the problem specification. Of course, the programs are written in a standard programming language, without any reference to knowledge or possibility. In the vast majority of cases, the programs in question do not enumerate and/or explore possible states or scenarios. Indeed, the program designer is typically unfamiliar with formal notions of knowledge. This being the case, what sense does it make to talk of Agent 1 in Example 1 "knowing" or "not knowing" that  $\text{Max} = c$ ? Can it make sense to say that the Agent "considers it possible that there may be more than four nodes in the system"? After all, we may be talking about a 10-line program. It has no soul. Does it have thoughts, doubts and mental states?

Since agents act based on their local information, a protocol designer must ensure that agents obtain the necessary information for a given task, and that this information is applied correctly. Using the information-based notion of knowledge, the designer can ascribe knowledge to an agent without requiring it to have a soul, feelings, and self-awareness. As seen in the CTM example, it is natural to think in terms of whether or not Agent 1 knows  $\text{Max} = c$  at any given point in a run of a protocol solving CTM. (A formal definition of knowledge will be provided in Section 2.) Suppose that a protocol is designed to solve CTM in networks that may have a variety of sizes. If Agent 1 does not start out with local information ensuring that there are no more than four nodes in the system, then from the point of view of an outside observer the agent can be thought of as "considering it possible" that there may be more than four nodes.

Even in a simple network as in Figure 1, the CTM problem can be posed in different models, which can differ in essential aspects. A solution to CTM in one model might not solve the problem in another model. Indeed, the rationale behind distinct solutions, as well as their details, may vary considerably. Are there common features shared by all solutions to CTM?

Interestingly, all solutions to CTM, in all models, share one property: Agent 1 must **know** that  $\text{Max} = c$  in order to print the value  $c$ . Indeed, the ability to print the answer in a protocol for CTM reduces to detecting when the Max value is known. Of course, once Agent 1 knows that  $\text{Max} = c$  it can safely print  $c$ . Hence, knowing that  $\text{Max} = c$  is not just necessary, but also a sufficient condition for printing  $c$ . The CTM problem shows that knowledge and attaining knowledge can be a central and crucial aspect of a standard distributed application.

The need to know  $\text{Max} = c$  in solving CTM suggests that we consider a natural question: *When does Agent 1 know that  $\text{Max} = c$ ?* The answer is less straightforward than we might initially expect.

What is known depends in a crucial way on the protocol that the agents are following. Thus, in the setting of Example 1, if the agents follow the bottom-up protocol, then Agent 1 knows the maximum once it receives a single message from Agent 2. Knowledge is also significantly affected by features of the model. In CTM, if there is an upper bound (say 100) on the possible initial values, then an agent that sees this value knows the maximum. Knowledge about the network topology and properties of communication play a role as well. For example, consider a model in which Agent 1 has a clock, and a single clock cycle suffices for a message to be delivered, digested, and acted upon. Suppose that the protocol is such that all agents start simultaneously at time 0 and an agent forwards a value towards Agent 1 only if this value is larger than any value it has previously sent. Then in the network of Figure 1 Agent 1 will receive a message with value 100 from Agent 2 at time 1, and no further messages. If Agent 1 knows that the diameter of the network is 3, it will not know the maximum upon receiving this message. However, without receiving any further messages, at time 3 Agent 1 *will* know that the maximum is 100; no larger value can be lurking in the system.

## 1.2 KoP and Coordination

The fact that  $\text{Max} = c$  is a necessary condition for printing  $c$  is an essential feature of the CTM problem. We have argued that, in fact,  $K_1(\text{Max} = c)$  is also a necessary condition for printing  $c$ , as the **KoP** would suggest. But this is just one instance. Let us briefly consider another example.

**Example 2.** *Consider a bank whose ATMs are designed in such a way that an ATM will dispense cash only to a customer whose account shows a sufficiently large positive balance. Along comes Alice, who has a large positive balance, and tries to obtain a modest sum from the ATM. On this day, however, the ATM is unable to communicate with the rest of the bank and it declines to pay Alice. Thus, despite the fact that Alice has good credit, the ATM frustrates her and denies her request. Apparently, given its specification, the ATM has no choice. Intuitively, in order to satisfy the credit restriction, the ATM needs to know that a customer has good credit before dispensing cash. If the ATM may pay a customer that is not known to have good credit, there will be possible scenarios in which the ATM will violate its specification, and pay a customer that does not have credit. Notice, however, that the specification said nothing about the ATM's knowledge. It only imposed a restriction on the ATM's action, based on the state of Alice's account.*  $\square$

Both the CTM problem and the ATM example are instances in which the **KoP** clearly applies. The intuitive argument for why the **KoP** should apply very broadly is straightforward. If  $\phi$  is a necessary condition for performing  $\alpha$ , and agent  $i$  ever performs  $\alpha$  without knowing  $\phi$ , then there should be a possible scenario that is indistinguishable to agent  $i$ , in which  $\phi$  does not hold. Since the two scenarios are indistinguishable, the agent can perform  $\alpha$  in the second scenario, and violate the requirement that  $\phi$  is a necessary condition. A formal statement and proof requires a definition of necessary conditions, knowledge, as well as capturing a sense in which an action at one point implies the same action at any other, indistinguishable, point. This will be done in Section 3.

Most tasks in distributed systems are described by way of a specification. Such specifications typically impose a variety of necessary conditions for actions. The **KoP** implies that even though such specifications often do not explicitly discuss the agents' knowledge, they do in fact impose knowledge preconditions. Observe that the **KoP** applies to a task regardless of the means that are used to implement it. Any engineer implementing a particular task will have to ensure that preconditions are known when actions are taken. This is true whether or not the engineer reasons explicitly in terms of knowledge, and it is true even if the engineer is not even aware of the knowledge terminology. (Normally, neither may be

the case, of course.) The need to satisfy the **KoP** suggests that the design of distributed implementations must involve at least two steps. One is to make sure that the required knowledge is made available to an agent who needs to performed a prescribed action, and the other is ensuring that the agent detect that it knows the required preconditions. This is quite different from common practice in engineering distributed implementations [28].

We remark that the **KoP** can be expected to hold in a variety of multi-agent settings well beyond the realm of distributed systems. Thus, for example, suppose that a jellyfish is naturally designed so that it will never sting its own flesh. By the **KoP**, the cell activating the sting at a given point needs to *know* that it is not stinging the jellyfish's body when it "fires" its sting. The jellyfish is thus designed with some form of a "friend or foe" mechanism that is used in the course of activating the sting. Various biological activities can similarly be considered in light of the **KoP**: How does the organism know that certain preconditions are met? Our last example will come from the social science arena. Suppose that a society designs a legal system, that is required to satisfy the constraint that only people who are guilty of a particular crime are ever put in jail for committing this crime. By the **KoP**, the judge (or jury) must *know* that the person committed this crime in order to send him to jail.

As discussed above, specifications impose preconditions. Typically, these conditions relate an action to facts about the world (e.g., the maximal value, or the customer's good credit). In many cases, however, actions of different agents need to be coordinated. Consider a variant of CTM in which in addition to Agent 1 printing the maximum, Agent 4 needs to perform an action (say print the same value or print the minimal value), but not before Agent 1 does. Then Agent 1 performing her action is a condition for 4's action. In particular, Agent 4 would need to know that Agent 1 has already come to know  $Max = c$  for some  $c$  before 4 acts. In some cases, the identity of actions performed needs to be coordinated.

For a final example, suppose that Alice should perform an action  $\alpha_A$  only if Bob performs an action  $\alpha_B$  at least 5 time steps earlier. Then she needs to know that Bob acted at least 5 steps before when she acts. Indeed, if  $\psi$  is a necessary condition for  $\alpha_B$ , then Alice must know that "Bob knew  $\psi$  at least 5 time steps ago" when she acts, since knowing  $\psi$  is a necessary condition for Bob's performing  $\alpha_B$  (see [4, 5]). As these examples illustrate, given **KoP**, coordination can give rise to nested knowledge.

Simple instances of the **KoP** are often quite straightforward. Ensuring and detecting  $K_1(Max = c)$  is often fairly intuitive, and it not justify the overhead involved in developing a theory of knowledge for multi-agent systems. However, satisfying statements involving nested knowledge in particular models of computation can quickly become nontrivial. For this, it is best to have a clear mathematical model of knowledge in multi-agent systems. The next section reviews the runs and systems model.

## 2 Modeling Knowledge Using Runs and Systems

We now review the runs and systems model of knowledge of [10, 13]. The interested reader should consult [10] for more details. A *global state* is an "instantaneous snapshot" of the system at a given time. Let  $\mathcal{G}$  denote a set of global states. Time will be identified with the natural numbers  $\mathbb{N} = \{0, 1, 2, \dots\}$  for ease of exposition. A *run* is a function  $r: \mathbb{N} \rightarrow \mathcal{G}$  associating a global state with each instant of time. Thus,  $r(0)$  is the run's initial state,  $r(1)$  is the next global state, and so on. A *system* is a set  $R$  of runs. The same global state can appear in different runs, and in some systems may even appear more than once in the same run.

A central notion in our framework is that of an agent's *local state*, whose role is to capture the agent's local information at a given point. The precise details of the local state depend on the application. It could be the complete contents of an agent's memory at the given instant, or the complete sequence of events

that it has observed so far. for example. The rule of thumb is that the local state should consist of the local information that the agent may use when deciding which actions to take. Thus, for example, if agents are finite-state machines, it is often natural to identify an agent's local state with the automaton state that it is in. Formally, we assume that every global state determines a unique **local state** for each agent. We denote agent  $i$ 's local state in the global state  $r(t)$  by  $r_i(t)$ . Moreover, a global state with  $n$  agents  $\mathbf{A} = \{1, \dots, n\}$  will have the form  $r(t) = \langle r_e(t), r_1(t), \dots, r_n(t) \rangle$ , where  $r_e(t)$  is called the local state of the *environment*, and will serve to represent all aspects of the global state that are not included in the agents' local states. For example, it could represent messages in transit, the current topology of the network including what links may be down, etc.

## 2.1 Syntax and Semantics

We are interested in a propositional logic of knowledge, in which propositional facts and epistemic facts can be expressed. Facts will be considered to be true or false at a point  $(r, t)$ , with respect to a system  $R$ . More formally, given a set  $\Phi$  of primitive propositions and a set  $\mathbb{P} = \{1, \dots, n\}$  of the agents in the system, we define a propositional language  $\mathcal{L}_n^K(\Phi)$  by closing  $\Phi$  under negation ' $\neg$ ' and conjunction ' $\wedge$ ', as well as under knowledge operators  $K_i$  for all  $i \in \mathbb{P}$  (see [14]). Thus, for example, if  $p, q \in \Phi$  are primitive propositions and  $i, j \in \mathbb{P}$  are agents, then  $\neg K_i p \wedge K_j K_i \neg K_j q$  is a formula in  $\mathcal{L}_n^K(\Phi)$ . We typically omit the set  $\Phi$  and call  $\mathcal{L}_n^K$  the language for knowledge with  $n$  agents.

In a multi-agent system facts about the world, as well as the knowledge that agents have, can change dynamically from one time point to the next. We thus consider the truth of formulas of  $\mathcal{L}_n^K$  at *points* of a system  $R$ , where a point is a pair  $(r, t) \in R \times \mathbb{N}$ , and it is used to refer to time  $t$  in the run  $r$ . We denote the set of points of a system  $R$  by  $\text{Pts}(R) \triangleq R \times \mathbb{N}$ . Points will play the role of states of a Kripke structure.

The set  $\Phi$  of primitive propositions used in the analysis of any given multi-agent system  $R$  will depend on the application. Their truth at the points of the system needs to be explicitly defined. This is done by an *interpretation*  $\pi : \Phi \times \text{Pts}(R) \rightarrow \{T, F\}$ , where  $\pi(q, (r, t)) = T$  means that the proposition  $q$  holds at  $(r, t)$ . Formally, an *interpreted system* w.r.t. a set  $\Phi$  of primitive propositions is a pair  $(R, \pi)$  consisting of the system  $R$  and interpretation  $\pi$  for  $\Phi$  over  $\text{Pts}(R)$ . Just as we typically omit explicit reference to  $\Phi$ , we shall omit  $\pi$  as well, when this is unambiguous.

We assume from here on that the environment's state  $r_e(t)$  in a global state  $r(t)$  contains a "*history*" component  $h$  that records all actions taken by all agents at times  $0, 1, \dots, t-1$ . Formally, we take  $h$  to be a set of triples  $\langle \alpha, i, t' \rangle$ , which grows monotonically in time. An action  $\alpha$  is considered to be performed by  $i$  at the point  $(r, t)$  if and only if the triple  $\langle \alpha, i, t \rangle$ , denoting that action  $\alpha$  was performed by agent  $i$  at time  $t$ , appears in the history component  $h$  of  $r_e(t')$  for all times  $t' > t$ .<sup>1</sup> For the analysis in this paper, we will also assume that  $\Phi$  includes propositions of the form  $\text{does}_i(\alpha)$  and  $\text{did}_i(\alpha)$  for agents  $i \in \mathbb{P}$  and actions  $\alpha$ . With this assumption, what actions are performed at any given point  $(r, t)$  is uniquely determined by the run  $r$ .

We will consider interpretations  $\pi$  that, on these propositions, are defined by

$$\pi(\text{does}_i(\alpha), (r, t)) = T \quad \text{iff} \quad \text{agent } i \text{ performs } \alpha \text{ at } (r, t)$$

$$\pi(\text{did}_i(\alpha), (r, t)) = T \quad \text{iff} \quad \pi(\text{does}_i(\alpha), (r, t')) = T \text{ holds for some } t' \leq t$$

We allow  $t' = t$  in the definition of  $\text{did}_i(\alpha)$  for technical convenience; it simplifies our later analysis slightly.

---

<sup>1</sup>Our definition does not imply or assume that the actions are observed, observable or recorded by any of the agents. Whether that is the case depends on the application.

Our model of knowledge will follow the standard Kripke-style possible worlds approach. The possibility relations that we use are induced directly from the system  $R$  being analyzed; two points are considered indistinguishable to an agent if its local states at the two points are the same. More formally:

**Definition 2.1.** *If  $r_i(t) = r'_i(t')$ , then  $(r, t)$  and  $(r', t')$  are called **indistinguishable to  $i$** , denoted by  $(r, t) \approx_i (r', t')$ .*

Formulae of  $\mathcal{L}_n^K$  are interpreted at a point  $(r, t)$  of an interpreted system  $(R, \pi)$  by means of the satisfaction relation ' $\models$ ', which is defined inductively by:

$$(R, r, t) \models p \text{ iff } (r, t) \in \pi(p);$$

$$(R, r, t) \models \neg\phi \text{ iff } (R, r, t) \not\models \phi;$$

$$(R, r, t) \models \phi \wedge \psi \text{ iff both } (R, r, t) \models \phi \text{ and } (R, r, t) \models \psi;$$

$$(R, r, t) \models K_i\phi \text{ iff } (R, r', t') \models \phi \text{ for all } (r', t') \in \text{Pts}(R) \text{ such that } (r', t') \approx_i (r, t).$$

We say that  $\phi$  is **valid in** the system  $R$ , and write  $R \models \phi$ , if  $(R, r, t) \models \phi$  for all points  $(r, t) \in \text{Pts}(R)$ . We say that  $\phi$  **validly implies**  $\psi$  **in**  $R$  if  $\phi \Rightarrow \psi$  is valid in  $R$ . Since, by Definition 2.1, the  $\approx_i$  relations are equivalence relations, each knowledge operator  $K_i$  satisfies the S5 axiom system [14]. In particular, it satisfies the **knowledge property** (or Knowledge Axiom) that  $K_i\phi \Rightarrow \phi$  is valid in all systems.

It is instructive to relate our modeling using runs and systems to standard multi-agent Kripke structures. For every system  $R$  there is a corresponding Kripke structure  $M_R = (S_R, \pi, \sim_1, \dots, \sim_n)$  for  $n$  agents such that  $S_R = \text{Pts}(R)$  and ' $\sim_i$ ' = ' $\approx_i$ ' for every  $i$ . They correspond in that  $(R, r, t) \models \phi$  iff  $M_R, (r, t) \models \phi$  is guaranteed for all  $(r, t) \in \text{Pts}(R) = S_R$  and  $\phi \in \mathcal{L}_n^K(\Phi)$  (for details, see [10]).

The system  $R$  will determine the space of possible runs and possible points, which play a crucial role in determining the truth of facts involving knowledge. For example, consider a run  $r$  in which Alice sends Bob a message at time 1, and Bob receives it at time 2. If  $R$  is a system in which messages may be lost, or may take longer than one time step to be delivered, then Alice would not know at time 2 (i.e., w.r.t.  $(R, r, 2)$ ) that her message has been delivered, because there is another run  $r' \in R$  that she cannot tell apart from  $r$  at time 2, in which her message is not (or not yet) delivered by that time. The same run  $r$  also belongs to another system  $R'$  in which messages are always reliably delivered in exactly one round. With respect to  $(R', r, 2)$ , however, Alice *would* know at time 2 that her message has been delivered.

Our definition of knowledge is rather flexible and widely applicable. The set  $R$  of the possible runs immediately induces what the agents know. Observe that the definition of knowledge is completely external. It ascribes knowledge to agents in the system even if the protocol they follow, as well as the actions that they perform, do not involve the knowledge terminology in any way. Moreover, the agents do not need to be complex or sophisticated for the definition to apply. Indeed, in a model of a very simple system consisting of a bed lamp and its electric cable, a switch in the OFF state can be said to know that the lamp is not lit; what the same switch would know in the ON state would depend on the system  $R$  under consideration, which determines the runs considered possible. E.g., if  $R$  contains a run in which the lamp is burnt out, then in the ON state the switch would not know that the lamp is shining light. On the other hand, if the lamp can never burn out, and the cord, plug and switch are in proper working order in all runs of  $R$ , then in the ON state the switch *would* know that the lamp is shining light. As this example shows, knowledge under this definition does not require the “knower” to compute what it knows. Indeed, this definition of knowledge is not sensitive to the computational complexity of determining what is known. In most cases, of course, we will ascribe knowledge to agents or components that can perform actions, which is not the case in the light switch example. And agents might need to explicitly establish whether they know relevant facts. We now provide a statement and proof of the knowledge of preconditions principle **KoP**.

### 3 Formalizing the Knowledge of Preconditions Principle

Intuitively, the **KoP** states that if a particular fact  $\psi$  is a necessary condition for an agent to perform an action  $\alpha$ , then the agent must in fact *know*  $\psi$  in order to act. In other words, *knowing*  $\psi$  is also a necessary condition for performing the action. We formalize the claim and prove it as follows. We say that  $\psi$  is a **necessary condition for**  $\text{does}_i(\alpha)$  **in**  $R$  if  $(R, r, t) \models \text{does}_i(\alpha)$  holds only if  $(R, r, t) \models \psi$ , for all  $(r, t) \in \text{Pts}(R)$ . Clearly, the customer's good credit is a necessary condition for the ATM dispensing cash. That is, suppose that a bank makes use of a correct implementation of an ATM protocol, which satisfies the credit requirement. Then, in the system  $R$  consisting of the set of all possible histories (runs) of the bank's (and the ATM's) transactions, good credit is a necessary condition for receiving cash from the ATM.

It is often of interest to consider facts whose truth depends only on a given agent's local state. Such, for example, may be the receipt of a message, or the observation of a signal, by the agent. Whether  $x = 0$  for a local variable  $x$ , for example, would be a natural local fact. Moreover, if an agent has perfect recall, then any events that it has observed in the past will give rise to local facts. Finally, since knowledge is defined based on an agent's local state, then a fact of the form  $K_i\phi$  constitutes a local fact. Indeed, there is a simple way to define the local facts above using knowledge. Namely, we say that  $\phi$  is **i-local in**  $R$  if  $R \models (\phi \Rightarrow K_i\phi)$ .

The formalism of [10] defines protocols as explicit objects, and defines *contexts* that describe the possible initial states and the model of computation. This provides a convenient and modular way of constructing systems. Namely, given a protocol  $P$  and a *context*  $\gamma$ , the system  $R = R(P, \gamma)$  is defined to be the set of all runs of protocol  $P$  in  $\gamma$ . The runs of this system embody all of the properties of the context, as they arise in runs of  $P$ . This includes, for example, any timing assumptions, possible values encountered, possible topologies of the network, etc. They also embody the relevant properties of the protocol, because in all runs considered possible the agents follow  $P$ .

In this paper, we do not define protocols and contexts. Rather, we treat the **KoP** in a slightly simpler and more abstract setting. We say that an action  $\alpha$  is a **conscious action for**  $i$  **in**  $R$  if  $i$ 's local state completely determines whether  $i$  performs  $\alpha$ . If its local state at two points  $(r, t)$  and  $(r', t')$  of  $R$  is the same, then  $(R, r, t) \models \text{does}_i(\alpha)$  iff  $(R, r', t') \models \text{does}_i(\alpha)$ . Conscious actions are quite prevalent in many systems of interest. For example, suppose that agent  $i$  follows a deterministic protocol, so that its action at any given point is a function of its local state. If, in addition, agent  $i$  is allowed to move at every time step, then all of its actions are conscious actions. We remark that, since conscious actions depend on an agent's local state, then if  $\alpha$  is conscious for  $i$  in  $R$  then  $(R, r, t) \models \text{does}_i(\alpha)$  holds iff  $(R, r, t) \models K_i\text{does}_i(\alpha)$  does, for all  $(r, t) \in \text{Pts}(R)$ .

We are now ready to prove a formal version of the **KoP**:

**Theorem 3.1** (The **KoP** Theorem). *Let  $\alpha$  be a conscious action for  $i$  in  $R$ . If  $\psi$  is a necessary condition for  $\text{does}_i(\alpha)$  in  $R$ , then  $K_i\psi$  is also a necessary condition for  $\text{does}_i(\alpha)$  in  $R$ .*

*Proof.* We will show the contrapositive. Let  $\alpha$  be a conscious action for  $i$  in  $R$ , and assume that  $K_i\psi$  is not a necessary condition for  $\text{does}_i(\alpha)$  in  $R$ . Namely, there exists a point  $(r, t) \in \text{Pts}(R)$  such that both  $(R, r, t) \models \text{does}_i(\alpha)$  and  $(R, r, t) \not\models K_i\psi$ . Given the latter, we have by the definition of ' $\models$ ' for  $K_i$  that there exists a point  $(r', t') \in \text{Pts}(R)$  such that both  $(r', t') \approx_i (r, t)$  and  $(R, r', t') \not\models \psi$ . Since  $\alpha$  is a conscious action for  $i$  in  $R$  and  $(R, r, t) \models \text{does}_i(\alpha)$  we have that  $(R, r, t) \models K_i\text{does}_i(\alpha)$ . It follows from  $(r', t') \approx_i (r, t)$  by the definition of ' $\models$ ' for  $K_i$  that  $(R, r', t') \models \text{does}_i(\alpha)$  holds. But since  $(R, r', t') \not\models \psi$ , we conclude that  $\psi$  is not a necessary condition for  $\text{does}_i(\alpha)$  in  $R$ , establishing the contrapositive claim.  $\square$



Theorem 3.1 applies to all multi-agent systems. It immediately implies, for example, that a necessary condition for the ATM to dispense cash is  $K_{atm}(\text{good\_credit})$ . The theorem is model independent; it does not depend on timing assumptions, on the topology of the system (even on whether agents communicate by message passing or via reading and writing to registers in a shared memory), or on the nature of the activity that is carried out. For every necessary condition for a conscious action, *knowing* that the condition holds is also a necessary condition.

## 4 Coordinating Simultaneous Actions

Recall that the language  $\mathcal{L}_n^K$  contains formulas in which knowledge operators can be *nested* to arbitrary finite depth. It is sometimes useful to consider a state of knowledge called *common knowledge* that goes beyond any particular nested formula. Intuitively, a fact  $\psi$  is common knowledge if everyone knowing that everyone knows  $\dots$ , that everyone knows the fact  $\psi$ , to every finite depth. Common knowledge has a number of equivalent definitions, one of which is as follows:

**Definition 4.1** (Common Knowledge). *Fix a set of agents  $G$  and a fact  $\psi$ . We denote by  $C_G\psi$  the fact that  $\psi$  is common knowledge to  $G$ . Its truth at points of a system  $R$  is defined by:*

$$(R, r, t) \models C_G\psi \quad \text{iff} \quad (R, r, t) \models K_{i_1}K_{i_2}\dots K_{i_m}\psi \text{ holds for all } \langle i_1, i_2, \dots, i_m \rangle \in G^m \text{ and all } m \geq 1.$$

Common knowledge, a term coined by Lewis in [18], plays an important role in the analysis of games [2], distributed systems [13], and many other multi-agent settings. Clearly, common knowledge is much stronger than “plain” knowledge. Indeed,  $C_G\psi$  validly implies  $K_j\psi$ , for all agents  $j \in G$ . Since common knowledge requires infinitely many facts to hold, it is not *a priori* obvious that  $C_G\phi$  can be attained at a reasonable cost, or even whether it can ever be attained at all, in settings of interest (see [7, 10, 13]). We will now show that there are natural applications for which attaining common knowledge is essential.

Intuitively, distinct actions are simultaneous in  $R$  if they can only be performed together; whenever one is performed, all of them are performed simultaneously. It is possible to define simultaneous coordination formally in terms of necessary conditions:

**Definition 4.2** (Simultaneous Actions). *Let  $G$  be a set of agents. We say that a set of actions  $A = \{\alpha_i\}_{i \in G}$  is (necessarily) **simultaneous in  $R$**  if  $\text{does}_i(\alpha_i)$  is a necessary condition for  $\text{does}_j(\alpha_j)$  in  $R$ , for all  $i, j \in G$ .*

Suppose that the actions in  $A$  are simultaneous in  $R$  in the above sense. Then the **KoP** immediately implies (by Theorem 3.1) that a necessary condition for performing an action in  $A$  is knowing that the other actions are also (currently) being performed. In fact, however, much more must be true. We now present a strong variant of the **KoP**, which shows that in order to perform simultaneous actions agents must attain common knowledge of their necessary conditions. Notice that in order to allow a set of actions by the agents in  $G$  to be simultaneous, the system  $R$  must be sufficiently deterministic to ensure that if  $i, j \in G$  are distinct agents and  $(R, r, t) \models \text{does}_i(\alpha)$  holds, then  $j$  will be scheduled to perform an action at  $(r, t)$ . For otherwise, there would be no way to ensure simultaneous execution of the actions by the agents in  $G$ . Conscious actions fit this setting well in this case. We proceed as follows.

**Theorem 4.3** (C-K of Preconditions). *Let  $G$  be a set of agents and let  $A = \{\alpha_i\}_{i \in G}$  be a set of necessarily simultaneous actions in the system  $R$ . Moreover, suppose that each action  $\alpha_i \in A$  is a conscious action for its agent  $i$  in  $R$ . If  $\psi$  is a necessary condition for  $\text{does}_i(\alpha_i)$  for some  $i \in G$ , then  $C_G\psi$  is a necessary condition for  $\text{does}_j(\alpha_j)$ , for all  $j \in G$ .*

*Proof.* Assume that  $A$  is a set of necessarily simultaneous actions for  $G$  in  $R$ . It is straightforward to show the following claim.

**Observation 1.** *Let  $\alpha_i, \alpha_j \in A$  be the actions for agents  $i$  and  $j$ , respectively. If a fact  $\varphi$  is a necessary condition for  $\text{does}_i(\alpha_i)$  in  $R$  then  $\varphi$  is also a necessary condition for  $\text{does}_j(\alpha_j)$  in  $R$ .*

To prove this observation notice that, by assumption, both (a)  $R \models \text{does}_j(\alpha_j) \Rightarrow \text{does}_i(\alpha_i)$  and (b)  $R \models \text{does}_i(\alpha_i) \Rightarrow \varphi$  hold. For all  $(r, t) \in \text{Pts}(R)$ , if  $(R, r, t) \models \text{does}_j(\alpha_j)$  then  $(R, r, t) \models \text{does}_i(\alpha_i)$  holds by (a) and so  $(R, r, t) \models \varphi$  by (b). Thus,  $\varphi$  is a necessary condition for  $\text{does}_j(\alpha_j)$  in  $R$ .

Assume that  $\psi$  is a necessary condition for  $\text{does}_i(\alpha_i)$ , for some  $i \in G$ . We shall prove by induction on  $m \geq 0$  that  $K_{i_1} K_{i_2} \cdots K_{i_m} \psi$  is a necessary condition for  $\text{does}_j(\alpha_j)$  in  $R$ , for every  $j \in G$  and all sequences  $\langle i_1, \dots, i_m \rangle \in G^m$  (of  $m$  agent names from  $G$ ). This will establish that  $(R, r, t) \models \text{does}_j(\alpha_j)$  implies  $(R, r, t) \models C_G \psi$  for all  $(r, t) \in \text{Pts}(R)$ , and thus  $C_G \psi$  is a necessary condition for  $\text{does}_j(\alpha_j)$  for all  $j \in G$ , as claimed.

- Base case: Let  $m = 0$ . The claim in this case is that if  $\psi$  is a necessary condition for  $\text{does}_i(\alpha_i)$  then  $\psi$  is also a necessary condition for  $\text{does}_j(\alpha_j)$ . This is precisely Observation 1, with  $\varphi \triangleq \psi$ .
- Inductive step: Let  $m \geq 1$ , and assume that the claim holds for all  $j' \in G$  and all sequences in  $G^{m-1}$ . Fix  $j \in G$  and a sequence  $\langle i_1, i_2, \dots, i_m \rangle \in G^m$ . Its suffix  $\langle i_2, \dots, i_m \rangle$  is a sequence in  $G^{m-1}$ . Thus,  $K_{i_2} \cdots K_{i_m} \psi$  is a necessary condition for  $\text{does}_{i_1}(\alpha_{i_1})$  by the inductive hypothesis for  $m - 1$  (applied to  $G^{m-1}$  and agent  $j' = i_1 \in G$ ). Given that  $\alpha_{i_1}$  is a conscious action by  $i_1$ , we can apply Theorem 3.1 to the necessary condition  $K_{i_2} \cdots K_{i_m} \psi$  and obtain that  $K_{i_1} K_{i_2} \cdots K_{i_m} \psi$  is a necessary condition for  $\text{does}_{i_1}(\alpha_{i_1})$ . By Observation 1 we have that  $K_{i_1} K_{i_2} \cdots K_{i_m} \psi$  is also a necessary condition for  $\text{does}_j(\alpha_j)$  in  $R$ , and we are done.

□

## 4.1 Common Knowledge and the Firing Squad Problem

As an illustration of the applicability of Theorem 4.3 to a concrete application, consider a simple version of the *Firing Squad* problem. In this instance, the set of agents  $G$  in the system must simultaneously perform an action (say each agent  $i \in G$  should perform the action  $\text{fire}_i$ ) in response to the receipt, by any agent in  $G$ , of a particular external input called a ‘go’ message. The  $\text{fire}_i$  action can stand for a simultaneous change in shared copies of a database, a public announcement at different sites of the system, or any other actions that need to take place simultaneously. Moreover,  $\text{fire}_i$  actions are allowed only if they are preceded by such a go message. For simplicity, we consider a case in which none of the agents in  $G$  may fail, and they all must satisfy the specification.

Let  $\psi_{\text{go}}$  be a proposition that is true at  $(r, t) \in \text{Pts}(R)$  if a go message is received by any of the agents in  $G$  at a point  $(r, t')$  of  $r$  at a time  $t' \leq t$ . According to the specification of the Firing Squad problem,  $\psi_{\text{go}}$  is a necessary condition for the  $\text{fire}_i$  actions. An immediate consequence of Theorem 4.3 is:

**Corollary 4.4.**  *$C_G \psi_{\text{go}}$  is a necessary condition for all  $\text{fire}_i$  actions in the Firing Squad problem.*

Given Corollary 4.4, any solution to the firing squad problem must first attain common knowledge that a go message has been received. It is well-known (see [11, 13]) that common knowledge of a fact is observed simultaneously at all agents it involves. Suppose that every  $i \in G$  performs  $\text{fire}_i$  when  $C_G \psi_{\text{go}}$  first holds. Since all agents in  $G$  will come to know that  $C_G \psi_{\text{go}}$  immediately, they will fire simultaneously, as required by the problem specification. Indeed, Theorem 4.3 shows that this is the first time at which they *can* perform according to a correct protocol. Implementing simultaneous tasks such as the Firing

Squad therefore inherently involves, and often reduces to, ensuring and detecting  $C_G\psi_{go}$ . Recall that depending on the properties of the system, attaining such common knowledge might be impossible in some cases, or it might incur a substantial cost in others. Just as in the case of the **KoP**, this necessity is not due to our formalism. It is only exposed by our analysis. In every protocol that implements such a task correctly, the firing actions cannot be performed unless  $C_G\psi_{go}$  is attained.

There is an extensive literature on using common knowledge to obtain optimal protocols for simultaneous tasks [8, 9, 16, 20, 21, 23, 24, 26, 25]. Typically, they involve an explicit proof that common knowledge of a particular fact is a necessary condition for performing a set  $A$  of necessarily simultaneous actions. Theorem 4.3 or a variant of it suited for fault-tolerant systems can be used to establish this result in all of these cases. Moreover, one of the main insights from the analysis of [13] and of [11] is that when simultaneous actions are performed, the participating agents have common knowledge that they are being performed. Theorem 4.3 is a strict generalization of this fact.

## 5 Temporally Ordering Actions

So far, we have seen two essential connections between knowledge and coordinated action: performing actions requires knowledge of their necessary conditions, and performing simultaneous actions requires common knowledge of their necessary conditions. We now further extend the connection between states of knowledge and coordination, by showing that temporally ordering actions depends on attaining nested knowledge of necessary conditions. Following [5], we define temporally ordered actions:

**Definition 5.1** (Ben Zvi and Moses). *A sequence of actions  $\langle \alpha_1, \dots, \alpha_k \rangle$  (for agents  $1, \dots, k$ , respectively) is (linearly) **ordered in**  $R$  if  $\text{did}_{j-1}(\alpha_{j-1})$  is a necessary condition for  $\text{does}_j(\alpha_j)$  in  $R$ .*

Observe that this definition does not force an action  $\alpha_j$  to occur in a run in which  $\alpha_{j-1}$  occurs. Rather, if an action  $\alpha_j$  is performed in a given run, then it must be preceded by all actions  $\alpha_1, \dots, \alpha_{j-1}$ . Moreover, if we denote the time at which an action  $\alpha_i$  is performed in a run  $r$  by  $t_i$ , then we require that  $t_{j-1} \leq t_j$  for every action  $\alpha_j$  performed in  $r$ .

**Claim 1.** *Assume that the sequence  $\langle \alpha_1, \dots, \alpha_k \rangle$  is ordered in  $R$ . Then  $R \models (\text{did}_j(\alpha_j) \Rightarrow \text{did}_{j-1}(\alpha_{j-1}))$  for all  $2 \leq j \leq k$ .*

*Proof.* Assume that  $(R, r, t) \models \text{did}_j(\alpha_j)$ . Then, by definition of  $\text{did}_j(\alpha_j)$ , we have  $(R, r, \hat{t}) \models \text{does}_j(\alpha_j)$  for some  $\hat{t} \leq t$ . The fact that  $\langle \alpha_1, \dots, \alpha_k \rangle$  is ordered in  $R$  implies that  $\text{did}_{j-1}(\alpha_{j-1})$  is a necessary condition for  $\text{does}_j(\alpha_j)$  in the system  $R$ , and so  $(R, r, \hat{t}) \models \text{did}_{j-1}(\alpha_{j-1})$ . Since  $\text{did}_{j-1}(\alpha_{j-1})$  is a stable fact and  $t \geq \hat{t}$ , we obtain that  $(R, r, t) \models \text{did}_{j-1}(\alpha_{j-1})$ . The claim follows.  $\square$

We say that a fact  $\phi$  is **stable in**  $R$  if once true,  $\phi$  remains true. Formally, if  $(R, r, t) \models \phi$  and  $t' > t$  then  $(R, r, t') \models \phi$ , for all  $r \in R$  and  $t, t' \geq 0$ . Notice that while  $\text{does}_i(\alpha)$  is, in general, not a stable fact,  $\text{did}_i(\alpha)$  is always stable.

**Definition 5.2.** *We say that **agent  $i$  recalls  $\psi$  in**  $R$  if the fact  $K_i\psi$  is stable in  $R$ .*

The notion of *perfect recall*, capturing the assumption that agents remember all events that they take part in, is popular in the analysis of games and multi-agent systems [10, 29]. While perfect recall is a nontrivial assumption often requiring significant storage costs, selective recall of single facts such as  $\text{does}_j(\alpha_j)$  is a much weaker assumption, that can be assumed of a system  $R$  essentially without loss of generality. By adding a single bit to Agent  $j$ 's local state, whose value is 0 as long as  $j$  has not performed  $\alpha_j$  and 1 once the action has been performed, we can obtain a system  $R'$  that is isomorphic to  $R$ , in which Agent  $j$  recalls  $\text{does}_j(\alpha_j)$ .

**Claim 2.** Assume that  $\alpha_j$  is a conscious action for  $j$  in  $R$ , and that  $j$  recalls  $\text{did}_j(\alpha_j)$  in  $R$ . Then  $\text{did}_j(\alpha_j)$  is a  $j$ -local fact in  $R$ .

*Proof.* Suppose that  $(R, r, t) \models \text{did}_j(\alpha_j)$ . Then, by definition of  $\text{did}_j(\alpha_j)$ , we have  $(R, r, \hat{t}) \models \text{does}_j(\alpha_j)$  for some  $\hat{t} \leq t$ . Choose an arbitrary  $(r', t') \in \text{Pts}(R)$  satisfying that  $(r', t') \approx_j (r, \hat{t})$ . It follows that  $(R, r', t') \models \text{does}_j(\alpha_j)$  since  $\alpha_j$  is a conscious action for  $j$  in  $R$ . By definition of  $\text{did}_j(\alpha_j)$  it follows that  $(R, r', t') \models \text{did}_j(\alpha_j)$ . Now, by definition of  $\models$  for  $K_j$  we have that  $(R, r, \hat{t}) \models K_j \text{did}_j(\alpha_j)$ . By assumption,  $j$  recalls  $\text{did}_j(\alpha_j)$  in  $R$ , and so  $K_j \text{did}_j(\alpha_j)$  is stable in  $R$ . Thus, since  $t \geq \hat{t}$ , we obtain that  $(R, r, t) \models K_j \text{did}_j(\alpha_j)$ , as claimed.  $\square$

We can now show:

**Theorem 5.3** (Ordering and Nested Knowledge). Assume that

- the actions  $\langle \alpha_1, \dots, \alpha_k \rangle$  are ordered in  $R$ ,
- each agent  $j = 1, \dots, k$  recalls  $\text{did}_j(\alpha_j)$  in  $R$ ,
- $\alpha_j$  is a conscious action for  $j$  in  $R$ , for all  $j = 1, \dots, k$ , and
- $\psi$  is a stable necessary condition for the first action  $\text{does}_1(\alpha_1)$  in  $R$

Then  $K_j K_{j-1} \dots K_1 \psi$  is a necessary condition for the  $j^{\text{th}}$  action  $\text{does}_j(\alpha_j)$  in  $R$ , for all  $j \leq k$ .

*Proof.* Assuming the conditions of the theorem, we will prove by induction on  $j \leq k$  that  $\text{did}_j(\alpha_j)$  validly implies  $K_j K_{j-1} \dots K_1 \psi$  in  $R$ . Since  $\text{does}_j(\alpha_j)$  validly implies  $\text{did}_j(\alpha_j)$  by definition of  $\text{did}_j(\alpha_j)$ , this will yield that  $K_j K_{j-1} \dots K_1 \psi$  is a necessary condition for  $\text{does}_j(\alpha_j)$  in  $R$ , as claimed. We proceed with the inductive argument.

- Base case  $j = 1$ : Assume that  $(R, r, t) \models \text{did}_1(\alpha_1)$ . Claim 2 implies that  $(R, r, t) \models K_1 \text{did}_1(\alpha_1)$ . Let  $(r', t') \in \text{Pts}(R)$  be an arbitrary point satisfying that  $(r', t') \approx_1 (r, t)$ . Then  $(R, r', t') \models \text{did}_1(\alpha_1)$  by the knowledge property. Thus,  $(R, r', \hat{t}) \models \text{does}_1(\alpha_1)$  holds for some  $\hat{t} \leq t'$ , and because  $\psi$  is a necessary condition for  $\text{does}_1(\alpha_1)$  in  $R$ , we obtain that  $(R, r, \hat{t}) \models \psi$ . Since  $\psi$  is stable and  $t' \geq \hat{t}$ , we have that  $(R, r', t') \models \psi$ . By choice of  $(r', t')$  we have that  $(R, r, t) \models K_1 \psi$ , as claimed.
- Inductive step: Let  $j > 1$  and assume that  $K_{j-1} \dots K_1 \psi$  is a necessary condition for  $\text{did}_{j-1}(\alpha_{j-1})$  in  $R$ . Moreover, let  $(R, r, t) \models \text{did}_j(\alpha_j)$ . Since  $\alpha_j$  is a conscious action for  $j$ , Claim 2 implies that  $(R, r, t) \models K_j \text{did}_j(\alpha_j)$ . Choose an arbitrary  $(r', t') \in \text{Pts}(R)$  satisfying that  $(r', t') \approx_j (r, t)$ . By definition of  $K_j$ , it follows that  $(R, r', t') \models \text{did}_j(\alpha_j)$ . By Claim 1, since the sequence  $\langle \alpha_1, \dots, \alpha_k \rangle$  is ordered in  $R$  and  $j > 1$  we have that  $(R, r', t') \models \text{did}_{j-1}(\alpha_{j-1})$ . We now apply the inductive hypothesis to obtain that  $(R, r', t') \models K_{j-1} \dots K_1 \psi$ . Finally, we obtain that  $(R, r, t) \models K_j K_{j-1} \dots K_1 \psi$  by choice of  $(r', t')$  and the definition of ' $\models$ ' for  $K_j$ . The claim now follows.  $\square$

A slightly more restricted version of Theorem 5.3 was proved in [5]. Rather than consider an arbitrary necessary condition for  $\alpha_1$ , they proved a version for the case in which the first action  $\alpha_1$  is triggered by an external input to agent 1. Technically, the proofs are quite similar.

Theorem 5.3 provides a necessary, but possibly not sufficient, condition for ordering actions in distributed systems. If agent  $j$  acts strictly later than when  $K_j K_{j-1} \dots K_1 \psi$  first holds, then it may be inappropriate for agent  $j + 1$  to act when it knows that the fact  $K_j K_{j-1} \dots K_1 \psi$  holds (i.e., when  $K_{j+1} K_j \dots K_1 \psi$  first holds). Nevertheless, Theorem 5.3 is often very useful because it can be used as a guide for efficiently, and sometimes even optimally, performing a sequence of ordered actions. Intuitively, suppose

that we have a protocol whose goal is to perform  $\langle \alpha_1, \dots, \alpha_k \rangle$  in response to an externally generated trigger  $\psi$  (such as the ‘go’ message in Firing Squad). In particular, assume that  $\psi$  is a necessary condition for  $\alpha_1$ . Keeping the communication aspects of this protocol fixed, an optimally fast solution would be for each agent  $j \leq k$  to perform  $\alpha_j$  when  $K_j K_{j-1} \dots K_1 \psi$  first holds. Let  $R$  be the set of runs of such a protocol with  $r \in R$ , and let  $t_j$  and  $t_{j-1}$  be the earliest times at which  $(R, r, t_j) \models K_j K_{j-1} \dots K_1 \psi$  and  $(R, r, t_{j-1}) \models K_{j-1} \dots K_1 \psi$  hold in a run  $r$ , respectively. The knowledge property guarantees that  $K_j K_{j-1} \dots K_1 \psi$  validly implies that  $K_{j-1} \dots K_1 \psi$  in  $R$ , and so  $t_j \geq t_{j-1}$ . Since, by assumption,  $\alpha_j$  is performed at time  $t_j$  and  $\alpha_{j-1}$  at  $t_{j-1}$ , we have that agents perform actions in linear temporal order, as required by Definition 5.1. Clearly, none of the actions can be performed any earlier, as Theorem 5.3 shows. We conclude that in time-efficient protocols, the nested knowledge formula presented by the theorem can be both necessary and sufficient. In this sense, Theorem 5.3 suggests a recipe for obtaining time-efficient solutions for ordering actions.

Just as Theorem 4.3 implies that common knowledge is a necessary condition for simultaneous actions, we now have by Theorem 5.3 that nested knowledge is a necessary condition for performing actions in linear temporal order. And just as there is an established literature on when common knowledge is and is not attainable and on how it may arise, there are results concerning the communication structure that underlies attaining nested knowledge. Indeed, in a seminal paper [7], Chandy and Misra showed that in asynchronous systems  $R$ , if  $(R, r, t) \models \neg \phi$  and at a time  $t' > t$   $(R, r, t') \models K_j K_{j-1} \dots K_1 \phi$ , then there must be a message chain in the run  $r$  between times  $t$  and  $t'$ , passing through the agents  $1, 2, \dots, j$  in this order (possibly involving additional agents as well). Given Theorem 5.3, this implies that the only way to coordinate actions in a linear temporal order in an asynchronous setting is by way of such message chains.<sup>2</sup>

More recently, Ben Zvi and Moses extended Chandy and Misra’s work to systems in which communication is not asynchronous, but rather agents may have access to clocks and the transmission time for each of the channels is bounded [5]. They show that a communication structure called a *centipede* must be constructed in order to obtain nested knowledge of spontaneous facts such as the arrival of an external input. They prove a slightly more restricted instance of Theorem 5.3 (without using **KoP** directly), and use it to show that ordering actions in their setting requires the construction of the appropriate centipedes. Finally, Parikh and Krasucki analyze the ability to create *levels of knowledge* consisting of collections of nested knowledge formulas in [27]. Theorem 5.3 relates levels of knowledge to coordination.

## 6 Discussion

This paper formulated the knowledge of preconditions principle and presented three theorems relating knowledge and coordinated action: the first is the **KoP** itself—necessary conditions for an action must be *known* to hold when the action is performed. Next, we showed that necessary conditions for simultaneous actions must be commonly known when the actions are taken. Finally, nested knowledge is a necessary condition for coordinating linearly ordered actions. The latter two are fairly direct consequences of the **KoP**. We discussed some of the uses of the latter two results in Sections 4 and 5. Indeed the **KoP** has many further implications.

In recent years, several works that make use of **KoP** have appeared, citing the unpublished [22]. For example, Castañeda, Gonczarowski and Moses used the **KoP** to analyze the consensus problem [6],

---

<sup>2</sup>Theorems 3.1 and 5.3 depend on conscious actions and therefore do not apply to asynchronous systems. Nevertheless, variants of these theorems can be presented that do apply to asynchronous systems and nondeterministic protocols. Details will appear in [22].

in which agents need to agree on a binary value in a fault-prone system. They designed a protocol in two steps—applying the **KoP** once to derive a rule by which, roughly, agents decide on 0 when they know of an initial value of 0. Then, they applied the **KoP** again assuming that this is the rule used for making decisions on 0, and obtained a rule involving nested knowledge (roughly, a statement of the form “knowing that nobody knows 0”) for deciding on the value 1. The result of their analysis was a very efficient solution to consensus that is optimal in a strong sense: It is the first unbeatable consensus protocol. No protocol can strictly dominate it, by having processes always decide at least as fast, and sometimes strictly faster, than this protocol does. The work of [6] complements an earlier work by Halpern, Moses and Waarts [15], in which a fixed point analysis of optimal consensus was obtained. The latter, too, is closely related to the **KoP**.

Gonczarowski and Moses used the **KoP** to analyze the epistemic requirements of more general forms of coordination [12]. Namely, they considered a setting in which  $k$  agents need to perform actions, and there are time bounds on the relative times at which the actions of any pair of agents is performed. The simple instance in which all bounds are 0 is precisely that of the simultaneous actions considered in Section 4. They show that such coordination requires vectorial fixed points of knowledge conditions, which are naturally related to fixed points and equilibria. The papers [3, 4, 5, 12] together can all be viewed as making use of the **KoP** to provide insights into the interaction between time and communication for coordinating actions in a distributed and multi-agent system. Describing them is beyond the scope of the current paper.

The most significant aspect of the **KoP**, in our view, is the fact that it places a new emphasis on the epistemic aspects of problem solving in a multi-agent system. Simple necessary conditions induce epistemic conditions. Thus, in order to act correctly, one needs a mechanism ensuring that the agents obtain the necessary knowledge, and that they discover that they have this knowledge. Most problems and solutions are not posed or described in this fashion. We believe that the **KoP** encapsulates an important connection between knowledge, action and coordination that will find many applications in the future.

## References

- [1] Hagit Attiya & Jennifer Welch (2004): *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, doi:10.1002/0471478210.
- [2] R. J. Aumann (1976): *Agreeing to disagree*. *Annals of Statistics* 4(6), pp. 1236–1239, doi:10.1214/aos/1176343654.
- [3] Ido Ben-Zvi & Yoram Moses (2013): *Agent-Time Epistemics and Coordination*. In: *Proceedings of ICLA*, pp. 97–108, doi:10.1007/978-3-642-36039-8\_9.
- [4] Ido Ben-Zvi & Yoram Moses (2013): *The Shape of Reactive Coordination Tasks*. In: *Proceedings of TARK, TARK XIV*, pp. 29–38.
- [5] Ido Ben-Zvi & Yoram Moses (2014): *Beyond Lamport’s Happened-before: On Time Bounds and the Ordering of Events in Distributed Systems*. *J. ACM* 61(2), p. 13, doi:10.1145/2542181.
- [6] Armando Castañeda, Yannai A Gonczarowski & Yoram Moses (2014): *Unbeatable Consensus*. In: *Proceedings of DISC*, Springer, pp. 91–106, doi:10.1007/978-3-662-45174-8\_7.
- [7] K. M. Chandy & J. Misra (1986): *How processes learn*. *Distributed Computing* 1(1), pp. 40–52, doi:10.1007/BF01843569.
- [8] Danny Dolev, Ezra N Hoch & Yoram Moses (2012): *An optimal self-stabilizing firing squad*. *SIAM Journal on Computing* 41(2), pp. 415–435, doi:10.1137/090776512.
- [9] C. Dwork & Y. Moses (1990): *Knowledge and common knowledge in a Byzantine environment: crash failures*. *Information and Computation* 88(2), pp. 156–186, doi:10.1016/0890-5401(90)90014-9.

- [10] R. Fagin, J. Y. Halpern, Y. Moses & M. Y. Vardi (2003): *Reasoning about Knowledge*. MIT Press, Cambridge, Mass.
- [11] Ronald Fagin, Joseph Y. Halpern, Yoram Moses & Vardi. Moshe Y. (1999): *Common knowledge revisited*. *Annals of Pure and Applied Logic* 96(13), doi:10.1016/S0168-0072(98)00033-5.
- [12] Y Gonczarowski & Y Moses (2013): *Timely common knowledge: Characterising asymmetric distributed coordination via vectorial fixed points*. In: *Proceedings of TARK XIV*.
- [13] J. Y. Halpern & Y. Moses (1990): *Knowledge and Common Knowledge in a Distributed Environment*. *Journal of the ACM* 37(3), pp. 549–587, doi:10.1145/800222.806735. A preliminary version appeared in *Proc. 3rd ACM PODC*, 1984.
- [14] J. Y. Halpern & Y. Moses (1992): *A guide to completeness and complexity for modal logics of knowledge and belief*. *Artificial Intelligence* 54, pp. 319–379, doi:10.1016/0004-3702(92)90049-4.
- [15] Joseph Y. Halpern, Yoram Moses & Orli Waarts (2001): *A Characterization of Eventual Byzantine Agreement*. *SIAM J. Comput.* 31(3), pp. 838–865, doi:10.1137/S0097539798340217.
- [16] Maurice P Herlihy, Yoram Moses & Mark R Tuttle (2011): *Transforming worst-case optimal solutions for simultaneous tasks into all-case optimal solutions*. In: *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, ACM, pp. 231–238, doi:10.1145/1993806.1993849.
- [17] Gérard Le Lann (1977): *Distributed Systems-Towards a Formal Approach*. In: *IFIP Congress*, 7, Toronto, pp. 155–160.
- [18] D. Lewis (1969): *Convention, A Philosophical Study*. Harvard University Press, Cambridge, Mass.
- [19] J. McCarthy & P. J. Hayes (1969): *Some Philosophical Problems From the Standpoint of Artificial Intelligence*. In: *Machine Intelligence 4*, Edinburgh University Press, pp. 463–502, doi:10.1016/b978-0-934613-03-3.50033-7.
- [20] Tal Mizrahi & Yoram Moses (2008): *Continuous consensus via common knowledge*. *Distributed Computing* 20(5), pp. 305–321, doi:10.1007/s00446-007-0049-6.
- [21] Tal Mizrahi & Yoram Moses (2008): *Continuous consensus with ambiguous failures*. In: *Distributed Computing and Networking*, Springer, pp. 73–85, doi:10.1016/j.tcs.2010.04.025.
- [22] Y Moses (2016): *Knowledge and Coordinated Action*, to appear.
- [23] Y. Moses & M. R. Tuttle (1988): *Programming simultaneous actions using common knowledge*. *Algorithmica* 3, pp. 121–169, doi:10.1007/BF01762112.
- [24] G. Neiger (1990): *Consistent coordination and continual common knowledge*. *Manuscript*.
- [25] G. Neiger & M. R. Tuttle (1993): *Common knowledge and consistent simultaneous coordination*. *Distributed Computing* 6(3), pp. 334–352, doi:10.1007/BF02242706.
- [26] Gil Neiger & Rida A Bazzi (1999): *Using knowledge to optimally achieve coordination in distributed systems*. *Theoretical computer science* 220(1), pp. 31–65, doi:10.1016/S0304-3975(98)00236-9.
- [27] R. Parikh & P. Krasucki (1992): *Levels of knowledge in distributed computing*. *Sādhana* 17(1), pp. 167–191, doi:10.1007/bf02811342.
- [28] Fred B Schneider (1990): *Implementing fault-tolerant services using the state machine approach: A tutorial*. *ACM Computing Surveys (CSUR)* 22(4), pp. 299–319, doi:10.1145/98163.98167.
- [29] R. Selten (1975): *Reexamination of the perfectness concept for equilibrium points in extensive games*. *International Journal of Game Theory* 4, pp. 25–55, doi:10.1145/2542181.